Tejas Nitin Joshi
#102181822

tejas.joshi@colorado.edu

ELEC 5623 RTES

LAB 0 REPORT

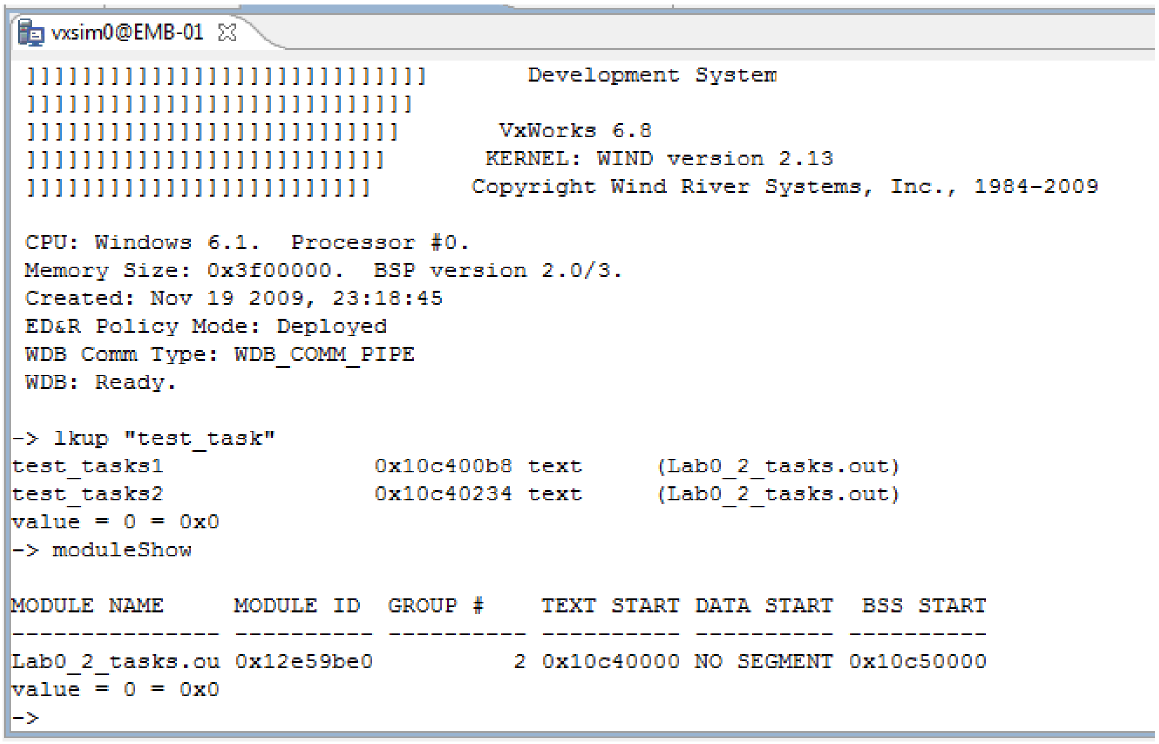
Familiarization with basic VxWorks tasking, cross compilation, and cross debugging

1.Cross Compiling for VxWorks

CROSS COMPILING AND DOWNLOADING THE KERNEL AND “TWO_TASKS.C”

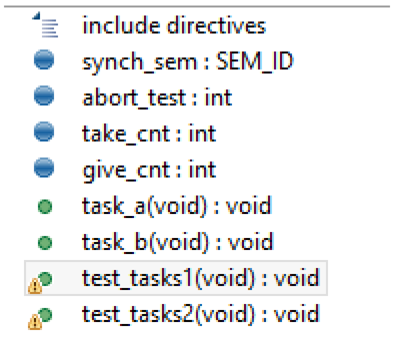
After cross compiling the VxWorks kernel and also building the “two_tasks.c” it was downloaded onto a target. Initially I had stared with the real target but was not able to continue because of the flood affecting the main server and completed the rest of the lab on the Simulator

Below is a screenshot for the moduleShow and lkup”test_task” commands on the terminal.

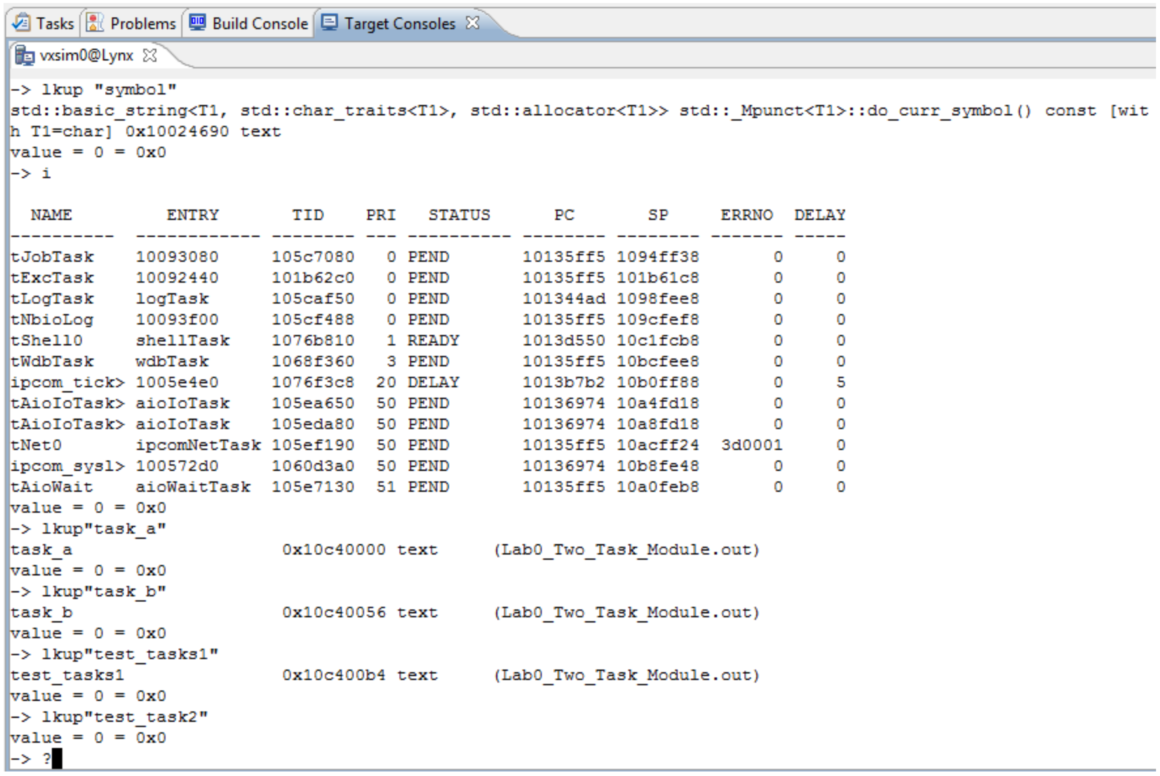


```
 ]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]       Development System
 ]]]]]]]]]]]]]]]]]]]]]]]]]]]]]
 ]]]]]]]]]]]]]]]]]]]]]]]]]]]]       VxWorks 6.8
 ]]]]]]]]]]]]]]]]]]]]]]]]]]]       KERNEL: WIND version 2.13
 ]]]]]]]]]]]]]]]]]]]]]]]]]]       Copyright Wind River Systems, Inc., 1984-2009

 CPU: Windows 6.1.  Processor #0.
 Memory Size: 0x3f00000.  BSP version 2.0/3.
 Created: Nov 19 2009, 23:18:45
 ED&R Policy Mode: Deployed
 WDB Comm Type: WDB_COMM_PIPE
 WDB: Ready.

-> lkup "test_task"
test_tasks1               0x10c400b8 text     (Lab0_2_tasks.out)
test_tasks2               0x10c40234 text     (Lab0_2_tasks.out)
value = 0 = 0x0
-> moduleShow

MODULE NAME     MODULE ID  GROUP #    TEXT START DATA START  BSS START
--------------- ---------- ---------- ---------- ---------- ----------
Lab0_2_tasks.ou 0x12e59be0          2 0x10c40000 NO SEGMENT 0x10c50000
value = 0 = 0x0
->
```

The entry point functions for the two_tasks.c are dynamically linked into kernel symbol table this can be viewed by using the lkup”symbol” command on the terminal screenshot as shown below

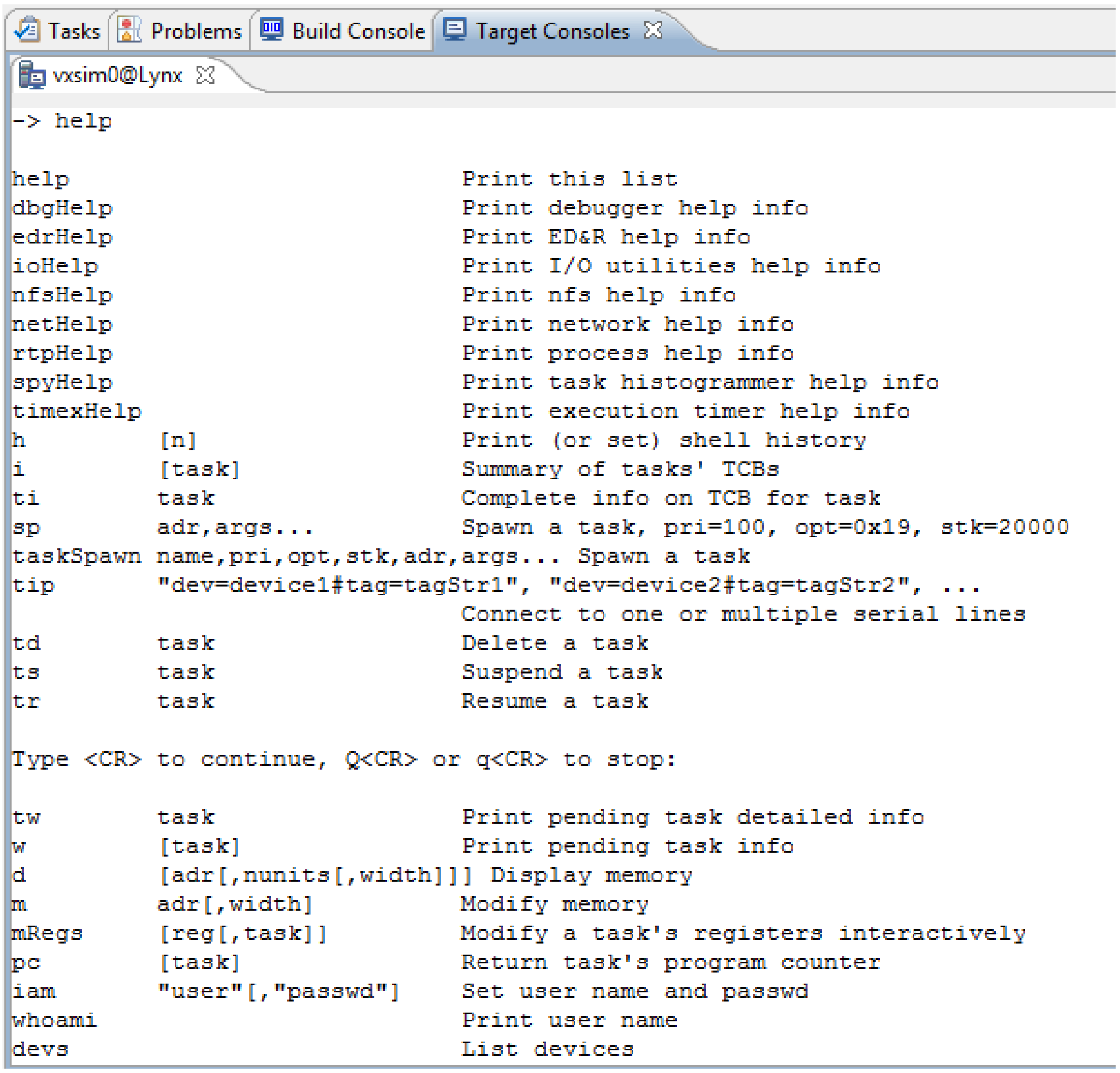


```
include directives
synch_sem : SEM_ID
abort_test : int
take_cnt : int
give_cnt : int
task_a(void) : void
task_b(void) : void
test_tasks1(void) : void
test_tasks2(void) : void
```



```
-> lkup "symbol"
std::basic_string<T1, std::char_traits<T1>, std::allocator<T1>> std::_Mpunct<T1>::do_curr_symbol() const [wit
h T1=char] 0x10024690 text
value = 0 = 0x0
-> i

  NAME         ENTRY        TID      PRI   STATUS      PC       SP      ERRNO   DELAY
----------  ------------ -------- --- ---------- -------- -------- ------- -----
tJobTask    10093080     105c7080   0 PEND       10135ff5 1094ff38       0     0
tExcTask    10092440     101b62c0   0 PEND       10135ff5 101b61c8       0     0
tLogTask    logTask      105caf50   0 PEND       101344ad 1098fee8       0     0
tNbioLog    10093f00     105cf488   0 PEND       10135ff5 109cfef8       0     0
tShell0     shellTask    1076b810   1 READY      1013d550 10c1fcb8       0     0
tWdbTask    wdbTask      1068f360   3 PEND       10135ff5 10bcfee8       0     0
ipcom_tick> 1005e4e0     1076f3c8  20 DELAY      1013b7b2 10b0ff88       0     5
tAioIoTask> aioIoTask    105ea650  50 PEND       10136974 10a4fd18       0     0
tAioIoTask> aioIoTask    105eda80  50 PEND       10136974 10a8fd18       0     0
tNet0       ipcomNetTask 105ef190  50 PEND       10135ff5 10acff24  3d0001     0
ipcom_sysl> 100572d0     1060d3a0  50 PEND       10136974 10b8fe48       0     0
tAioWait    aioWaitTask  105e7130  51 PEND       10135ff5 10a0feb8       0     0
value = 0 = 0x0
-> lkup"task_a"
task_a                    0x10c40000 text     (Lab0_Two_Task_Module.out)
value = 0 = 0x0
-> lkup"task_b"
task_b                    0x10c40056 text     (Lab0_Two_Task_Module.out)
value = 0 = 0x0
-> lkup"test_tasks1"
test_tasks1               0x10c400b4 text     (Lab0_Two_Task_Module.out)
value = 0 = 0x0
-> lkup"test_task2"
value = 0 = 0x0
-> ?
```

The help command when typed in prints out a list of possible command syntaxes and corresponding descriptions that can be executed on the VxWorks Win shell prompt.



```
-> help

help                           Print this list
dbgHelp                        Print debugger help info
edrHelp                        Print ED&R help info
ioHelp                         Print I/O utilities help info
nfsHelp                        Print nfs help info
netHelp                        Print network help info
rtpHelp                        Print process help info
spyHelp                        Print task histogrammer help info
timexHelp                      Print execution timer help info
h          [n]                 Print (or set) shell history
i          [task]              Summary of tasks' TCBs
ti         task                Complete info on TCB for task
sp         adr,args...         Spawn a task, pri=100, opt=0x19, stk=20000
taskSpawn name,pri,opt,stk,adr,args... Spawn a task
tip        "dev=device1#tag=tagStr1", "dev=device2#tag=tagStr2", ...
                               Connect to one or multiple serial lines
td         task                Delete a task
ts         task                Suspend a task
tr         task                Resume a task

Type <CR> to continue, Q<CR> or q<CR> to stop:

tw         task                Print pending task detailed info
w          [task]              Print pending task info
d          [adr[,nunits[,width]]] Display memory
m          adr[,width]         Modify memory
mRegs      [reg[,task]]        Modify a task's registers interactively
pc         [task]              Return task's program counter
iam        "user"[,"passwd"]   Set user name and passwd
whoami                         Print user name
devs                           List devices
```

2.two_task.c

CODE DESCRIPTION

The two_tasks.c file consists of initializing a variable “synch_sem” of type SEM_ID to store semaphore identification information.

Global int variables take_cnt and give_cnt store the number of SemTake() and SemGive() execution numbers.

task_a :

This routine adds a delay of 1000 ticks and then performs semGive and fills the synch_sem semaphore 10,000,000 times. It also increments the give_cnt every iteration.

task_b:

This routine takes away the semaphore in a loop of 10000000 times and similar to task a maintains a count. But the delay introduced here by the taskDelay () function is after.

Test_task1:

This routine deletes the tasks_a and task_b, provides corresponding error handling with.

The semBCreate () function creates a Binary Semaphore and assigns a priority here task a is assigned 100 and b is given 90 hence B preempts A.

Tesk_task2:

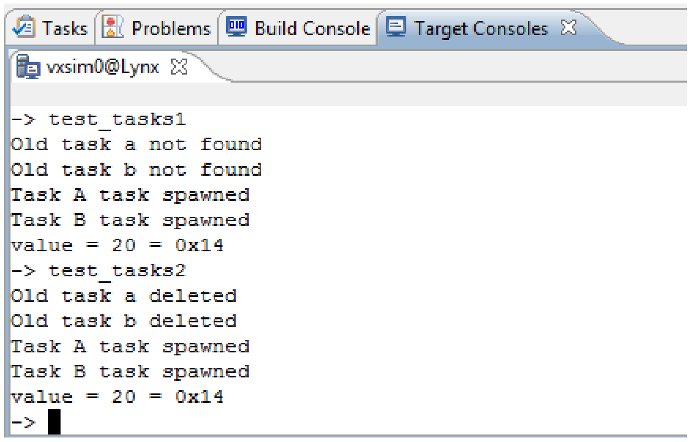
The deletion and re creation of tasks is same as test_task1 but priorities assigned are in a reverse fashion A =90 and B=100

SYNCHRONIZATION:

Test_task1() has an irregular priority setting hence there will be more likely pre-emption of task B (which takes away the semaphore) when the binary semaphore is empty.

Test_task2() has a good priority setting where semaphores will be made more quickly available to task_b and will run in a clockwork like sequence of gives and takes.

SCREENSHOTS :



```
-> test_tasks1
Old task a not found
Old task b not found
Task A task spawned
Task B task spawned
value = 20 = 0x14
-> test_tasks2
Old task a deleted
Old task b deleted
Task A task spawned
Task B task spawned
value = 20 = 0x14
->
```

This report template is complete with styles for a Table of

Contents and an Index. On the **References tab**, click the table or index you want to insert, and then choose the options you want.

The index field collects index entries specified by XE. To insert an index entry field, select the text to be indexed. On the **References tab**, in the Index group, click **Mark Entry**.

> Tip: You can also open the **Mark Index Entry** dialog box more quickly by pressing ALT+SHIFT+X. The dialog box stays open so that you can mark index entries..

In addition to producing reports, this template can be used to create proposals and reports.

Semaphore Types:

Apart from the binary Semaphore Used in this lab. Othr Semaphore creation types are

semCLib - counting semaphores

semMLib - mutual exclusion semaphores

semSmLib - shared memory semaphores